Architecture:

MVC (Model-View-Controller)

Divides an application into three main components:

* Model:
  + Manages the data and logic of the application
* View:
  + Responsible for the presentation and user interface
* Controller:
  + Handles the interaction between the Model and the View
* Promotes modularity and scalability of applications.

This structure helps us separate the different responsibilities of each part of our game. Additionally, dividing it in modules helps with maintenance and it makes it easier to collaborate between the team members. Finally, this structure helps prevent anti-patterns, such as Spaghetti Code, with a clear structure. We divide the code into these parts:

* **Model**: Manages the game data and logic, such as player status, scores, and interaction logic with environmental elements.
* **View**: Renders the game interface, such as the HUD (indicators for health, time, and comfort).
* **Controller**: Captures user input and translates it into commands for the model or view, such as interaction with driving controls and responses to AI events.

# Design Patterns Used:

**1. Singleton Pattern**

* **Principle**: The Singleton pattern ensures that a class has only **one instance** and provides a global point of access to that instance.
* **How It Works**: The class itself controls the instantiation process by storing its single instance and providing a static method to access it. This prevents other classes from creating new instances of the class.
* **Use Case**: Used for classes like GameManager and AudioManager to ensure that there is only one instance of these managers during the game's lifecycle.

**2. Observer Pattern**

* **Principle**: The Observer pattern defines a **one-to-many dependency** between objects, so that when one object (the subject) changes state, all its dependents (observers) are notified and updated automatically.
* **How It Works**: The subject maintains a list of observers and provides methods to add, remove, and notify them of changes. When the subject’s state changes, it calls a method (e.g., notify()) to update all registered observers.
* **Use Case**: Ideal for implementing the comfort bar and notifications of changes in the game state, allowing the view to update information when the model changes.

**3. Factory Method Pattern**

* **Principle**: The Factory Method pattern provides an interface for **creating objects** but allows subclasses to alter the type of objects that will be created.
* **How It Works**: Instead of calling a constructor directly, a method (the factory method) is used to create an object. This approach promotes loose coupling by abstracting the creation process and delegating it to subclasses or dedicated factory classes.
* **Use Case**: Useful for dynamically creating instances of obstacles and interactive objects during the game.

# UML

# User Stories

**1. As a player, I want to have a main menu when the game starts so I can easily access the options to start a game, view scores, and settings.**

* A main menu appears when the game starts.
* The main menu includes options to start a game, view scores, and adjust settings.
* The menu is accessible from any screen when the game is paused.

**2. As a taxi driver (player), I want to be able to drive my taxi precisely to pick up and drop off passengers at their destinations.**

* The taxi responds smoothly to movement controls (acceleration, braking, turning).
* The taxi has a health bar that decreases with collisions.
* The taxi has a comfort bar that reflects passenger comfort and affects tips.

**3. As a customer, I want the ride to be safe and comfortable so that I can give a good tip at the end.**

* The comfort bar fills or empties based on the quality of the ride.
* If comfort remains high, the player receives an extra tip at the end of the ride.
* Crashes or abrupt maneuvers reduce comfort and, therefore, potential tips.

**4. As a player, I want a TaxiApp in my taxi that lets me see destinations and estimated arrival times.**

* The TaxiApp displays the client’s current destination and the time limit to complete the ride.
* The TaxiApp has a clear interface on the HUD, updating remaining time in real-time.
* The TaxiApp offers basic navigation to guide the player to the destination.

**5. As a police officer, I want to be able to detect taxis that exceed the speed limit to initiate a pursuit.**

* The police automatically detect when the taxi exceeds speed limits.
* When detecting speeding, the police car starts a pursuit.
* The pursuit continues until the taxi slows down or evades the police.

**6. As a player, I want the taxi to be able to evade the police during a pursuit to avoid losing points or receiving penalties.**

* The player can lose the police by taking difficult routes or unexpected exits.
* When the police lose sight of the taxi, the pursuit ends.
* Successfully evading the police grants a points bonus.

**7. As a player, I want to find bonus items and obstacles around the city to improve my taxi or restore health.**

* There are visible bonus items in the city.
* Obstacles affect the taxi’s health bar.
* Speed or health bonuses apply upon collecting the corresponding item.

**8. As a customer, I want to arrive at my destination within a certain time to avoid losing tip points.**

* The customer has a time limit to reach the destination, visible on the taxi’s HUD.
* If the player arrives within the time limit, a time bonus is awarded.
* If the player does not arrive on time, the tip is reduced, or the ride is forfeited.

**9. As a taxi driver (player), I want to see a final score at the end of each ride to evaluate my performance and compare with previous scores.**

* At the end of each ride, a score is displayed based on time, comfort, and tips.
* Scores are recorded in a leaderboard accessible from the main menu.
* The highest score achieved is highlighted in the leaderboard.

**10. As a player, I want the game to have randomly generated characters and vehicles each session for greater variety and replayability.**

* Characters and vehicles in the city are randomly generated each session.
* Interactive objects and obstacles also change location in each game.
* The city layout may vary to add diversity in each session.

**11. As a player, I want to play in local multiplayer mode to compete or collaborate with friends in the same city.**

* The game allows at least two players in the same session in split-screen mode.
* Each player has their own taxi and can pick up clients and earn scores independently.
* Players can compete for tips or collaborate to achieve team goals.

**12. As a developer, I want the game to be designed in a modular way to allow new types of characters or vehicles to be added easily.**

* New characters can be integrated without modifying core logic.
* New vehicles can inherit from VehicleController without needing to rewrite the vehicle control system.
* Character and vehicle AI can be customized without impacting the game’s overall structure.

**13. As a developer, I want the game to log key events to identify performance issues or bugs.**

* Major events (collisions, pursuits, points earned) are recorded in logs.
* Logs are available for review at the end of each game session.
* Logs help identify potential failures in game mechanics or performance issues.

# General Objective Analysis

## Game Objective:

## 

The goal is to create a driving simulator where the player assumes the role of a taxi driver in a chaotic city, transporting customers safely and quickly to their destinations while evading police pursuits and collecting bonuses. The game should be immersive, with a dynamic city that offers variety in each playthrough, and it should include challenging components such as speed limits, delivery times, and obstacles.

## Requirements and Tasks:

**User Story 1: Main Menu**

* Objective: Create a main interface to start the game, access scores, and settings.
* Requirements:
  + Main menu interface with Play, Scores, and Settings options.
  + Clear transition between the menu and game start.
* Tasks:
  + Design and code the main menu user interface.
  + Implement navigation between menu options.
  + Set up events to start the game and access scores.

**User Story 2: Taxi Driving**

* Objective: Enable the player to control the taxi precisely.
* Requirements:
  + Taxi movement control (acceleration, brake, turn).
  + Health bar and passenger comfort bar.
* Tasks:
  + Configure the physics and control system for the taxi.
  + Implement a health bar that decreases on collision.
  + Implement a comfort bar that responds to driving quality.

**User Story 3: Customer Safety and Comfort**

* Objective: Ensure that driving quality affects customer satisfaction.
* Requirements:
  + Comfort bar adjusts based on driving quality.
  + Additional tips based on comfort.
* Tasks:
  + Program conditions that modify the comfort bar.
  + Create logic to calculate tips based on customer comfort.

**User Story 4: TaxiApp**

* Objective: Allow the player to know the customer’s destination and the trip time limit.
* Requirements:
  + TaxiApp module displaying destination and time limit.
  + TaxiApp information in HUD to guide the player.
* Tasks:
  + Code the TaxiApp with destination and timer logic.
  + Integrate TaxiApp in the game HUD to display information on screen.

**User Story 5: Police Detect Speeding**

* Objective: The police AI should detect and pursue taxis that exceed the speed limit.
* Requirements:
  + Police speed detection system.
  + Pursuit logic triggered by speeding.
* Tasks:
  + Code speed detection and trigger pursuit events.
  + Implement police AI to respond to speeding.

**User Story 6: Evading the Police**

* Objective: Allow the player to evade the police when being pursued.
* Requirements:
  + Police lose sight of the player when they take specific routes.
  + Bonus points if the player successfully evades the pursuit.
* Tasks:
  + Implement police evasion conditions and logic.
  + Set up scoring system for successful evasion bonuses.

**User Story 7: Bonus Items and Obstacles**

* Objective: Add variety and interaction with bonus items and obstacles.
* Requirements:
  + Bonus items to restore health and increase speed.
  + Obstacles like construction barriers or other cars.
* Tasks:
  + Design and program bonus items and their effects on the taxi.
  + Add logic to decrease taxi health if it collides with obstacles.

**User Story 8: Timely Arrival at Destination**

* Objective: Add time pressure to complete the ride.
* Requirements:
  + Timer on HUD to display time limit.
  + Tip adjustment if the ride is completed on time or delayed.
* Tasks:
  + Set up timer in TaxiApp or HUD.
  + Create logic for tip bonuses or reductions based on time.

**User Story 9: Final Score**

* Objective: Show a final score to the player to assess performance.
* Requirements:
  + Scoring based on time, comfort, and tips.
  + Scoreboard accessible from the main menu.
* Tasks:
  + Calculate the score at the end of each ride.
  + Implement a leaderboard and storage for scores.

**User Story 10: Random Generation of Characters and Vehicles**

* Objective: Increase replayability with randomly generated elements in each game.
* Requirements:
  + Random generation of characters and vehicles each playthrough.
  + Obstacles and bonus items with random positions.
* Tasks:
  + Code logic for random generation of elements.
  + Adjust positions and quantity of obstacles and characters for variety each game.

**User Story 11: Local Multiplayer Mode**

* Objective: Enable local multiplayer in split-screen mode.
* Requirements:
  + Allow two players to control independent taxis.
  + Individual scoring system for each player.
* Tasks:
  + Set up split-screen mode for two players.
  + Duplicate controls and HUD for each player.
  + Create individual scoring logic in multiplayer.

**User Story 12: Modular Design for New Characters and Vehicles**

* Objective: Ensure the design allows for adding new characters and vehicles.
* Requirements:
  + Modular architecture to integrate new classes of characters and vehicles without modifying core code.
* Tasks:
  + Structure classes and inheritance to support extensions.

**User Story 13: Key Event Logging**

* Objective: Record key events to facilitate troubleshooting and analysis.
* Requirements:
  + Logging system for collisions, pursuits, and other important events.
  + Accessible log storage for review and analysis.
* Tasks:
  + Program event logging at key points.
  + Implement an interface to review logs after each game.